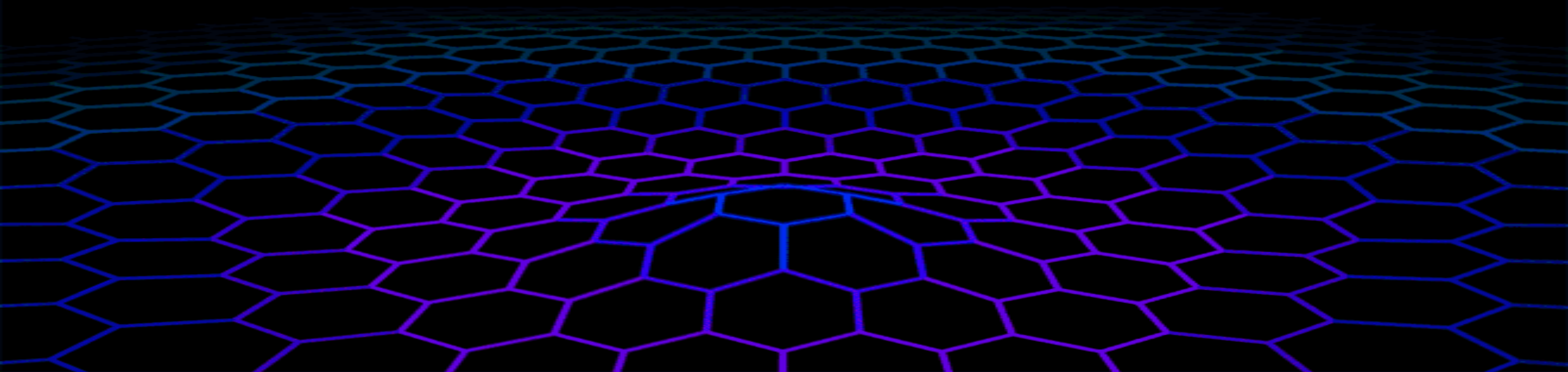
Unisenac
Campus Pelotas

Banco de Dados 2



Banco de Dados 2

STORED PROCEDURES (continuação)

Procedimentos (Procedures)

Repositório para um conjunto de declarações SQL.

Podem conter comandos de desvios condicionais e repetição como IF e WHILE.

São importantes para tarefas que devam ser realizadas de forma automatizada como atualizações que possam ser chamadas via SQL.

Possibilitam minimizar a quantidade de código fonte da aplicação e coloca estes códigos sob o controle da camada do banco de dados.

Faz com que o projeto da aplicação fique mais claro e deixa as páginas de código fonte da aplicação livres de códigos SQL dinâmicos.

Componentes da criação de Stored Procedures

Comandos comuns na manipulação de Stored Procedures.		
Comando	**Função**	**Exemplo**
CALL NOMEPROCEDURE	Realiza a execução de uma stored procedure através do console MySQL;	`CALL listaFilmes(12);`
SHOW PROCEDURE STATUS	Mostra as procedures que foram criadas no SGBD;	`SHOW PROCEDURE STATUS WHERE db = 'aula05';`
DELIMITER caractere	Define um novo delimitador de código, ou seja, o MySQL irá encerrar grupos de comandos através do caractere que for inserido.	`DELIMITER %`
CREATE PROCEDURE nomeProcedure	Realiza a criação de uma nova stored procedure, seu uso é obrigatório.	`CREATE PROCEDURE teste (num INT(3))`
DROP PROCEDURE nomeProcedure	Elimina a procedure do SGBD;	`DROP PROCEDURE teste;`

Componentes da criação de Stored Procedures

Comandos comuns na manipulação de Stored Procedures.		
Comando	**Função**	**Exemplo**

BEGIN – END	Define o corpo da stored procedure, o código da função será inserido entre os blocos BEGIN-END	`CREATE PROCEDURE teste(num INT(3))`
		`BEGIN`
		`    -- comandos`
		`END`

IF condicao
THEN
ELSE
	Cria um bloco com instruções IF. Podem ser utilizados comandos de seleção, inclusão, e demais comandos aceitos pela sintaxe MySQL.	`IF contador > 10 THEN`
		`   SELECT 'entrou no if';`
		`ELSE`
		`   SELECT 'entrou no else';`
		`END IF;`

END IF	Encerra um bloco completo IF, seu uso é obrigatório.	

Componentes da criação de Stored Procedures

Comandos comuns na manipulação de Stored Procedures.		
Comando	**Função**	**Exemplo**
DECLARE nomevariavel TIPO DEFAULT valor;	Realiza uma declaração de variável na stored procedure.	
SET nomevariavel = valor	Associa um novo valor a uma variável declarada;	
nomedoLoop: LOOP **Comandos** **END LOOP nomedoLoop;**	Define a inicialização de um bloco de repetição e finaliza o bloco;	
LEAVE nomedoLoop	Realiza a saída de um bloco de repetição;	
ITERATE nomedoLoop	Realiza o incremento de uma repetição;	

```
DELIMITER $$

DROP PROCEDURE IF EXISTS teste;
$$
CREATE PROCEDURE teste()
BEGIN
  DECLARE contador INT DEFAULT 0;
  meuPrimeiroLoop: LOOP
  SET contador = contador + 1;
  IF contador = 10 THEN
     SELECT 'Entrou no IF';
       LEAVE meuPrimeiroLoop;
  ELSE
     SELECT 'Entrou no ELSE - ',
contador;
    ITERATE meuPrimeiroLoop;
  END IF;
END LOOP meuPrimeiroLoop;
END
$$
DELIMITER ;

CALL teste();
```

Componentes da criação de Stored Procedures

Comandos comuns na manipulação de Stored Procedures.		
Comando	**Função**	**Exemplo**

DETERMINISTIC	Opcional. Usada com funções.
	
	Uma função é considerada "Determinística" se ela sempre retorna o mesmo resultado para os mesmos parâmetros de entrada, e "não determinística" caso contrário.
	
	Quando criamos uma function (CREATE FUNCTION) podemos especificar ou omitir a palavra DETERMINISTC.
	
	Quando especificada permite que a execução da SQL salve uma cópia do resultado de retorno da função para poder prover maior performance nas chamadas subsequentes com o mesmo valor de entrada na mesma SQL sem a necessidade de reexecutar a função.
	
	O otimizador escolhe a melhor performance entre obter a cópia dos resultados salva ou chamar novamente a função.
		`CREATE FUNCTION teste1()`
		`RETURNS INT`
		`DETERMINISTIC RETURN 4;`
		
		`SELECT teste1();`

Controlar estoque no MySQL
Usando somente triggers. (exemplo simples)

Controlar estoque no MySQL usando triggers

Controle de estoque de produtos é uma funcionalidade básica em sistemas desenvolvidos para o comércio e empresas em geral.

Controlar estoque no MySQL usando triggers

Permitir aos usuários consultar, em tempo real, a disponibilidade de um determinado produto.

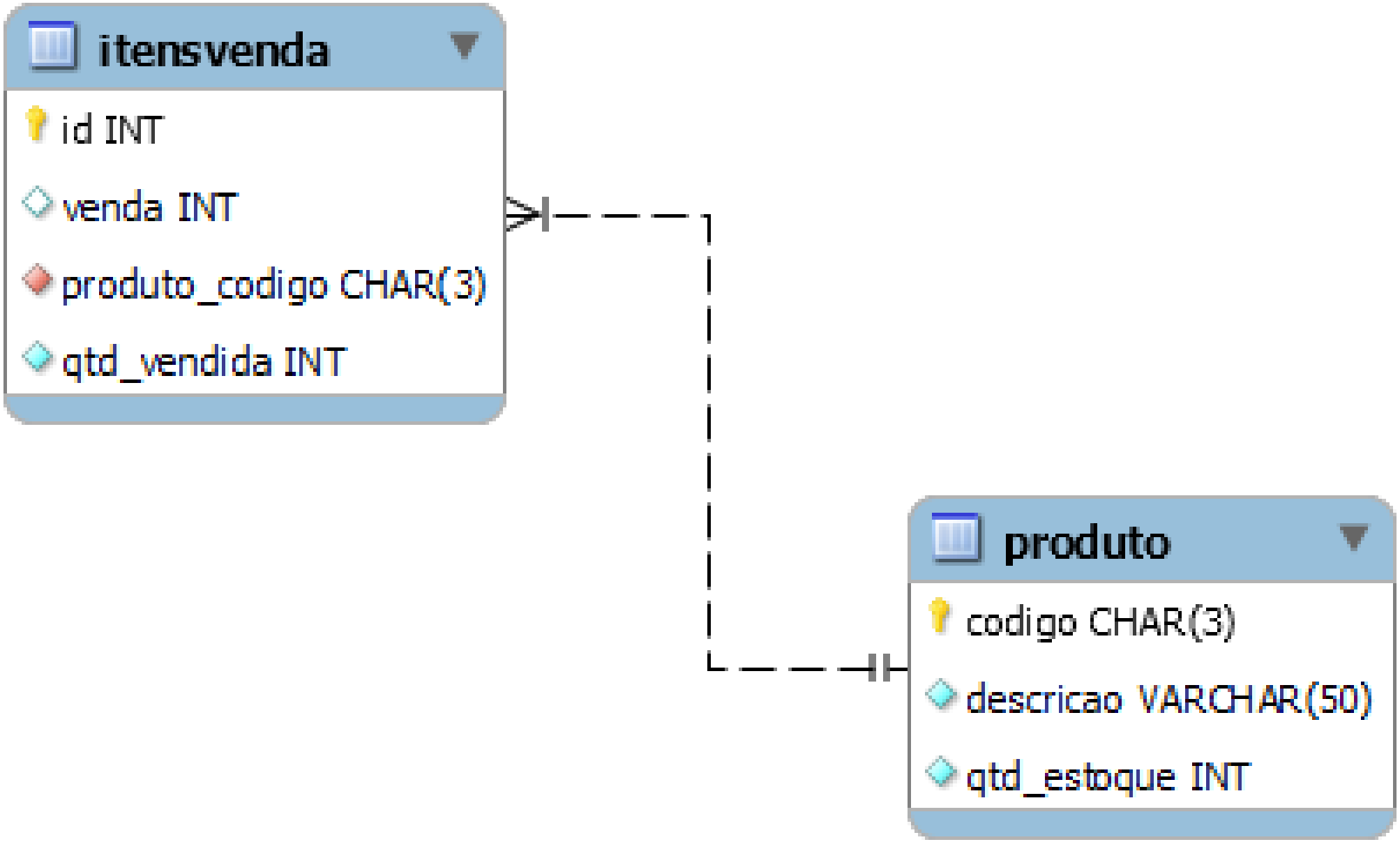
Exemplo

Um mercado que, ao realizar vendas, precisa que o estoque dos produtos seja automaticamente reduzido.

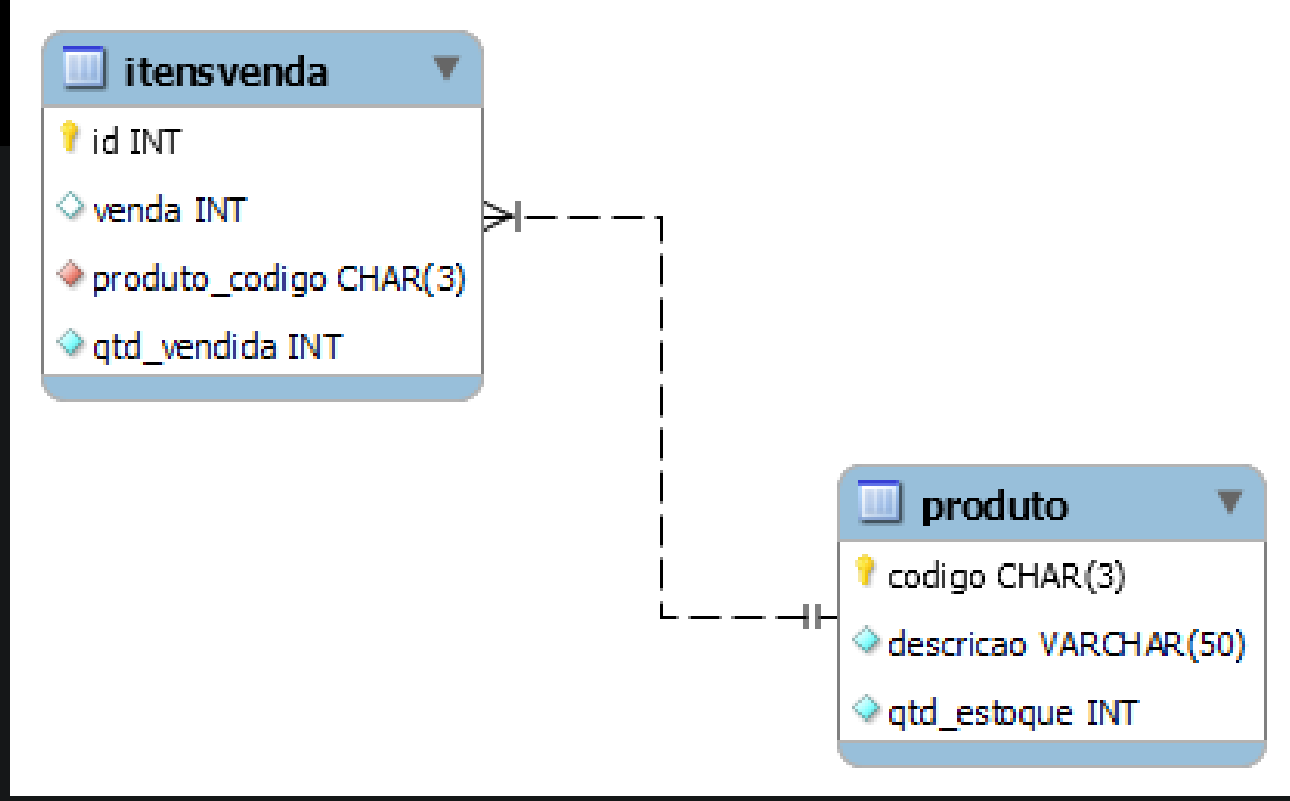
A devolução do estoque deve também ser automática no caso de remoção de produtos da venda.

```
-- Este exemplo não utiliza PROCEDURES, apenas TRIGGERS diretas nas tabelas.
```

Exemplo


itensvenda
id INT
venda INT
produto_codigo CHAR(3)
qtd_vendida INT
produto
codigo CHAR(3)
descricao VARCHAR(50)
qtd_estoque INT

Exemplo



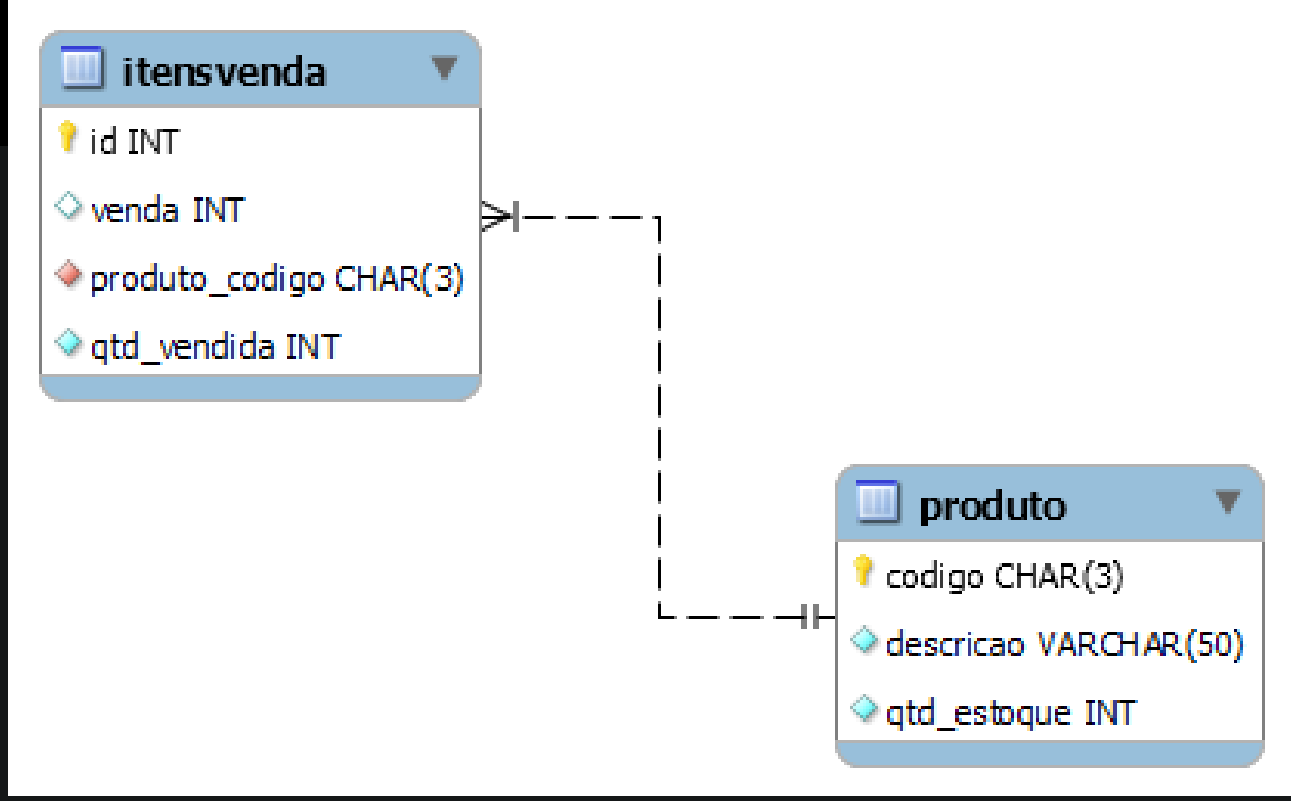

```sql
-- SLIDES AULA
DROP SCHEMA IF EXISTS aula05m;
CREATE SCHEMA IF NOT EXISTS aula05m;
USE aula05m;

-- Criando a tabela produto
DROP TABLE IF EXISTS produto;

CREATE TABLE IF NOT EXISTS produto (
  codigo      CHAR(3),
  descricao   VARCHAR(50) NOT NULL,
  qtd_estoque INT NOT NULL,
  CHECK (qtd_estoque > 0),
  PRIMARY KEY (codigo)
);

-- Inserindo na tabela produto
INSERT INTO produto (codigo, descricao, qtd_estoque) VALUES ('001', 'Feijão', 10);
INSERT INTO produto (codigo, descricao, qtd_estoque) VALUES ('002', 'Arroz', 5);
INSERT INTO produto (codigo, descricao, qtd_estoque) VALUES ('003', 'Farinha', 15);
SELECT * FROM produto ;
```

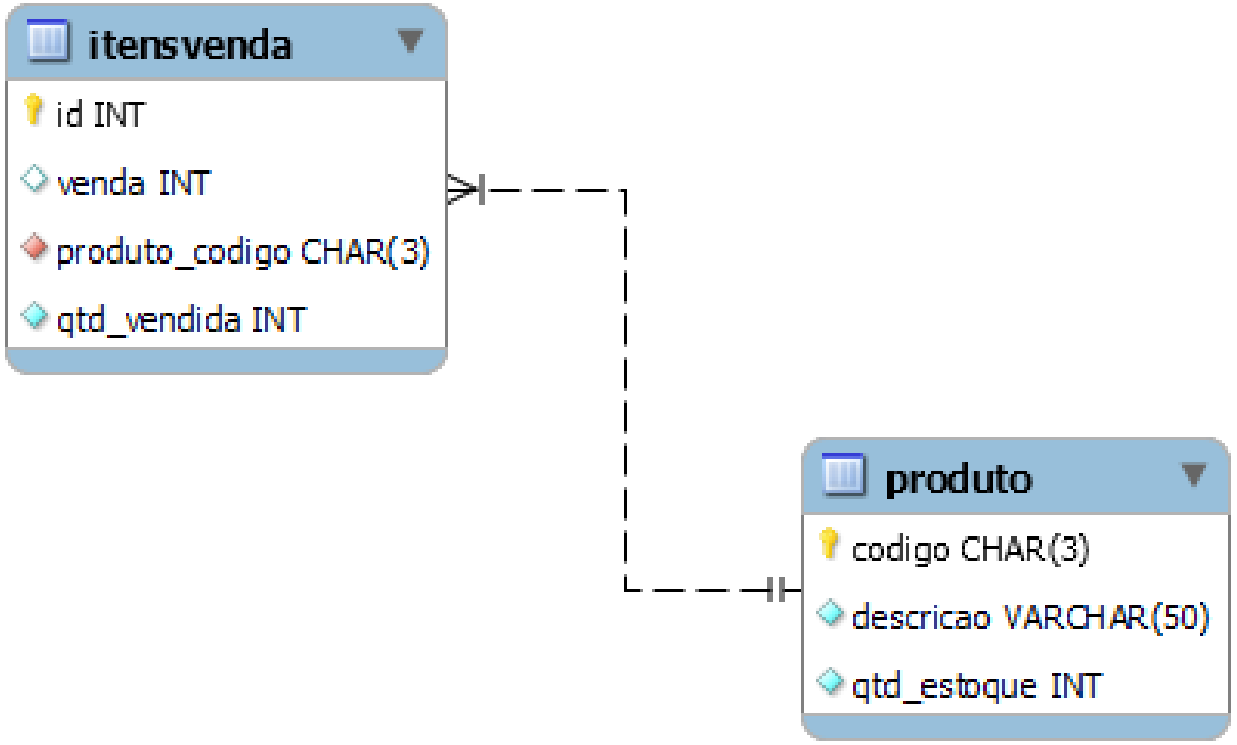
Exemplo




```sql
-- Criando a tabela itensVenda
DROP TABLE IF EXISTS itensVenda;

CREATE TABLE IF NOT EXISTS itensVenda (
  id              INT,
  venda           INT,
  produto_codigo CHAR(3) NOT NULL,
  qtd_vendida     INT(11) NOT NULL,
  PRIMARY KEY (id),
  CONSTRAINT fk_itensVenda_produto
    FOREIGN KEY (produto_codigo) REFERENCES produto (codigo)
);
```

Exemplo



Ao inserir e remover registro da tabela **itensVenda**, o estoque do produto referenciado deve ser alterado na tabela produto.

Para isso, serão criados dois triggers: um AFTER INSERT para dar baixa no estoque e um AFTER DELETE para fazer a devolução da quantidade do produto.

Exemplo

Como usaremos instruções que requerem ponto e vírgula no final, alteraremos o delimitador de instruções para $$ e depois de criar os triggers, voltaremos para o padrão.

Essa alteração não está diretamente ligada aos triggers.

Apenas para registrar e conferir, a imagem a seguir mostra um SELECT feito sobre a tabela produto após a inserção dos registros de exemplo.

```
SELECT * FROM produto;
+--------+-----------+-------------+
| codigo | descricao | qtd_estoque |
+--------+-----------+-------------+
| 001    | Feijão    |          10 |
| 002    | Arroz     |           5 |
| 003    | Farinha   |          15 |
+--------+-----------+-------------+
3 rows in set (0.00 sec)
```

Exemplo

Criação dos gatilhos(triggers) para executar as ações já discutidas.

```
-- Criando trigger que será executada após as inclusões
DELIMITER $$

CREATE TRIGGER trg_itensvenda_AI AFTER INSERT
ON itensVenda
FOR EACH ROW
BEGIN
    UPDATE produto SET produto.qtd_estoque = produto.qtd_estoque - NEW.qtd_vendida
    WHERE produto.codigo = NEW.produto_codigo;
END$$

DELIMITER ;
```

Foi utilizado o registro NEW para obter as informações da linha que está sendo inserida na tabela.

Exemplo

Criação dos gatilhos(triggers) para executar as ações já discutidas.

```
-- Criando trigger que será executada após as exclusões
DELIMITER $$

CREATE TRIGGER trg_itensvenda_AD AFTER DELETE
ON itensVenda
FOR EACH ROW
BEGIN
    UPDATE produto SET produto.qtd_estoque = produto.qtd_estoque + OLD.qtd_vendida
    WHERE produto.codigo = OLD.produto_codigo;
END$$

DELIMITER ;
```

Neste gatilho, se obtém os dados que estão sendo apagados da tabela através do registro OLD.

Exemplo

Tendo criado os triggers, podemos testá-los inserindo dados na tabela **itensVenda**.

```
-- Inserindo vendas
SELECT * FROM produto;
INSERT INTO itensVenda VALUES (1, 1, '003',2);
INSERT INTO itensVenda VALUES (2, 1, '001',3);
INSERT INTO itensVenda VALUES (3, 1, '002',1);
INSERT INTO itensVenda VALUES (4, 2, '002',1);
INSERT INTO itensVenda VALUES (5, 2, '003',4);
INSERT INTO itensVenda VALUES (6, 2, '001',3);
INSERT INTO itensVenda VALUES (7, 3, '001',1);
INSERT INTO itensVenda VALUES (8, 3, '002',2);
SELECT * FROM produto;
```

Exemplo

Nota-se que o estoque dos produtos foi corretamente reduzido, de acordo com as quantidades “vendidas”.

```
-- Antes de inserir uma venda
SELECT * FROM produto;
+--------+-----------+-------------+
| codigo | descricao | qtd_estoque |
+--------+-----------+-------------+
| 001    | Feijão    |          10 |
| 002    | Arroz     |           5 |
| 003    | Farinha   |          15 |
+--------+-----------+-------------+
3 rows in set (0.00 sec)
```

```
-- Após realizar uma venda
SELECT * FROM produto;
+--------+-----------+-------------+
| codigo | descricao | qtd_estoque |
+--------+-----------+-------------+
| 001    | Feijão    |           3 |
| 002    | Arroz     |           1 |
| 003    | Farinha   |           9 |
+--------+-----------+-------------+
3 rows in set (0.00 sec)
```

Exemplo

Agora para testar o trigger da exclusão, removeremos a primeira venda.

```
-- Excluindo vendas
DELETE FROM itensVenda WHERE id = 1;
```

Com isso, o estoque do produto ‘003’ (Farinha) deve ser acrescido de 2 e ficar com o valor 11.

```
-- Antes de excluir uma venda
SELECT * FROM produto;
+--------+-----------+-------------+
| codigo | descricao | qtd_estoque |
+--------+-----------+-------------+
| 001    | Feijão    |           3 |
| 002    | Arroz     |           1 |
| 003    | Farinha   |           9 |
+--------+-----------+-------------+
3 rows in set (0.00 sec)
```

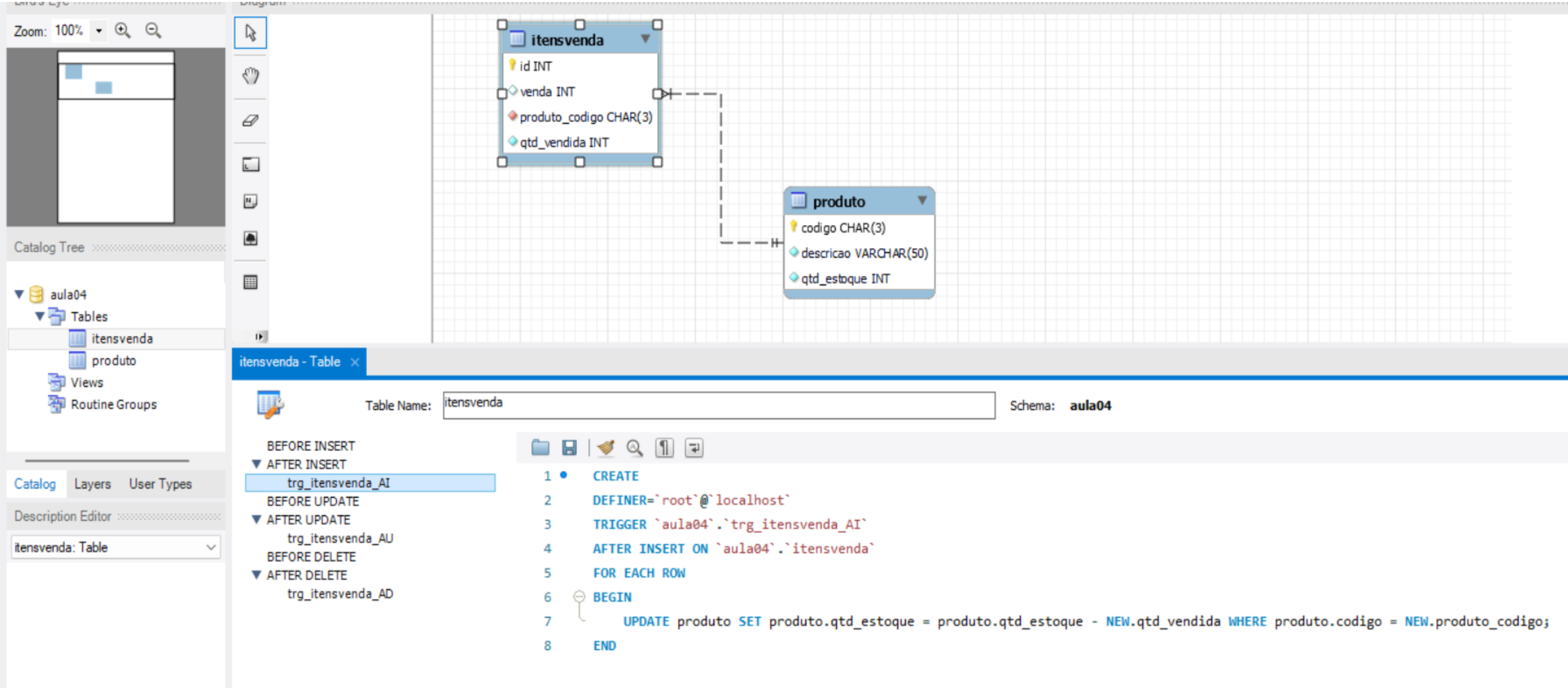
```
-- Após excluir uma venda
SELECT * FROM produto;
+--------+-----------+-------------+
| codigo | descricao | qtd_estoque |
+--------+-----------+-------------+
| 001    | Feijão    |           3 |
| 002    | Arroz     |           1 |
| 003    | Farinha   |          11 |
+--------+-----------+-------------+
3 rows in set (0.00 sec)
```

Controlar estoque no MySQL usando triggers

O estoque pode ser controlado através de várias técnicas. (escolha a sua)

MySQLWorkbench

No MySQLWorkbench é possível visualizar os gatilhos relacionados a uma tabela através do Object browser, como mostra a figura a seguir



Controlar estoque no MySQL usando triggers e stored procedures.

Criar um banco de dados chamado 'aula05bm' ('aula05bn')

Criar tabelas, **triggers** e **procedures** para controlar o estoque de uma pequena loja

TABELAS:

- produto
- produtoEntrada
- estoque
- produtoSaida

Controlar estoque no MySQL usando triggers e stored procedures.

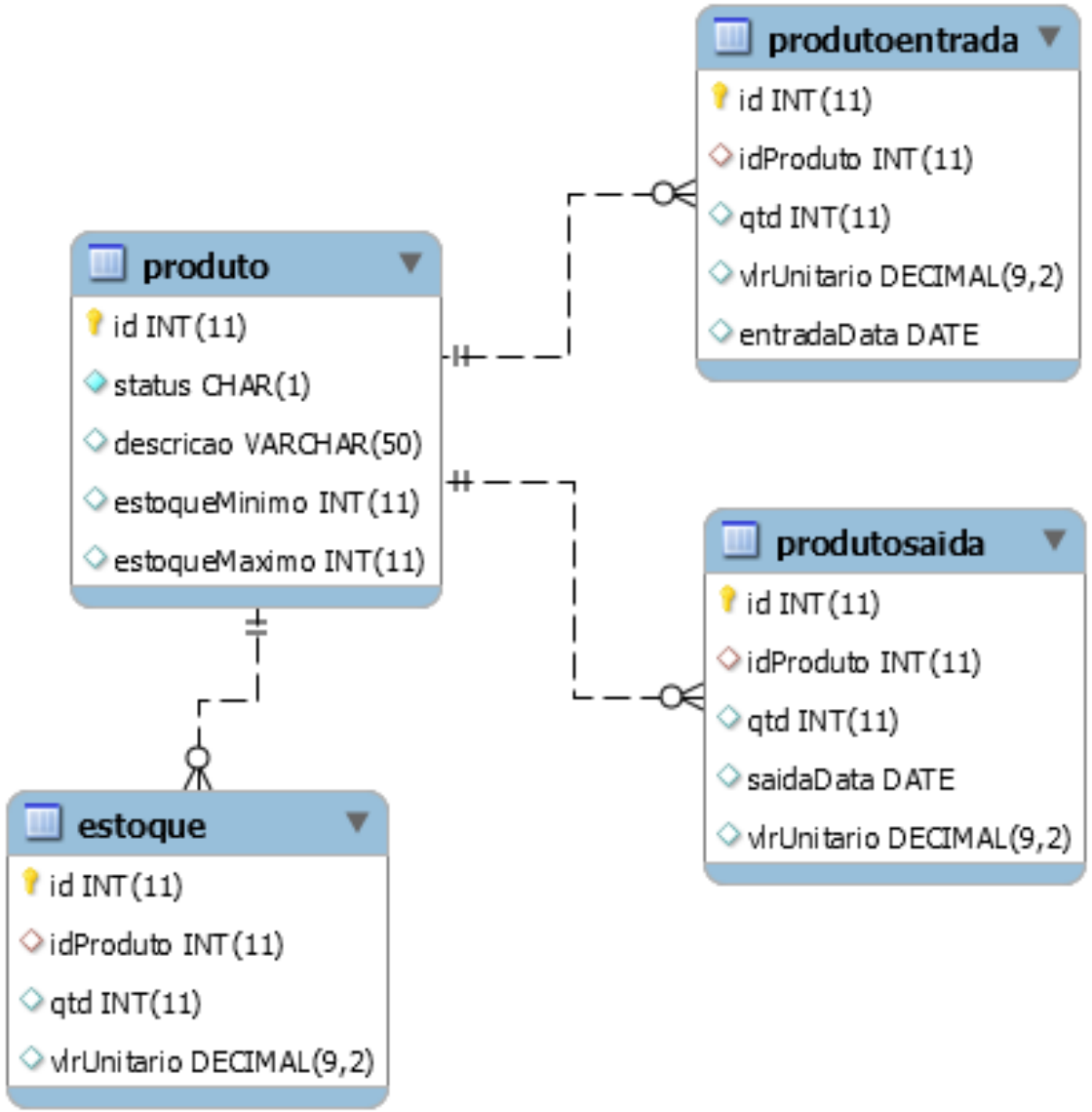


Tabela: **produto**

Armazena informações básicas sobre os produtos que a loja vende.

```sql
CREATE TABLE produto (
  id              INT AUTO_INCREMENT,
  status          CHAR(1) NOT NULL DEFAULT 'A',
  -- Indica se o cadastro está ativo 'A' ou inativo 'I'
  descricao       VARCHAR(50),
  estoqueMinimo INT(11),
  estoqueMaximo INT(11),
  PRIMARY KEY (id)
);
```

Tabela: **produto**

Cadastrando/Inserindo produtos na tabela.

```
INSERT INTO produto (descricao, estoqueMinimo, estoqueMaximo)
VALUES
('PENDRIVE', 10, 100),
('MOUSE',    10, 100),
('IOGURTE',   5,  50),
('TEQUILA',   5,  40),
('PRESUNTO',  5,  20);
```

Tabela: **estoque**

```sql
CREATE TABLE estoque (
  id          INT AUTO_INCREMENT,
  idProduto   INT,
  qtd         INT,
  vlrUnitario DECIMAL(9,2) NULL DEFAULT 0.00,
  PRIMARY KEY (id)
);
```

Tabela: **produtoEntrada** (compra)

Armazena as compras de produtos efetuadas para loja.

Obs.: através de ***triggers*** serão controladas inserções na tabela '**estoque**'

```
CREATE TABLE produtoEntrada (
  id              INT AUTO_INCREMENT,
  idProduto       INT,
  qtd             INT,
  vlrUnitario     DECIMAL(9,2) NULL DEFAULT 0.00,
  entradaData     DATE,
  PRIMARY KEY (id)
);
```

Tabela: **estoque**

- Recebe os dados conforme as ações executadas nas tabelas '**produtoEntrada**' e '**produtoSaida**'.

- O usuário não tem interação direta como INSERÇÕES, UPDATES e EXCLUSÕES,

- A tabela '**estoque**' armazena somente o resultado das ações de compra e venda de produtos.

Tabela: **produtoSaida** (venda)

Nessa tabela serão gravadas todas as saídas (Vendas) de produtos.
Obs.: através de **triggers** essas ações serão refletidas na tabela de '**estoque**'

```
CREATE TABLE produtoSaida (
  id           INT AUTO_INCREMENT,
  idProduto    INT,
  qtd          INT,
  saidaData    DATE,
  vlrUnitario DECIMAL(9,2) NULL DEFAULT 0.00,
  PRIMARY KEY (id));
```

Procedure: '**SP_AtualizaEstoque**'

Procedure para atualizar os estoques na tabela de '**estoque**'.

Nas quatro tabelas criadas existem dois campos em comum '**idProduto**' e '**qtd**', são estes campos que servirão como parâmetros para inserção e baixa de estoque nas *procedures*.

Procedure: '**SP_AtualizaEstoque**'

Recebe três parâmetros (**var_idProduto**, **var_qtdComprada**, **var_vlrUnitario**) e tem a finalidade de inserir ou debitar produtos na tabela de '**estoque**' de acordo com o os parâmetros que são passados.

```
Esta PROCEDURE é chamada pelas triggers para atualizar o estoque.
Ela verifica se já existe o produto no estoque:
- Se existir, atualiza a quantidade e o valor unitário.
- Se não existir, insere um novo registro.
```

```
DELIMITER $$
CREATE PROCEDURE SP_AtualizaEstoque (var_idProduto INT,
                                     var_qtdComprada INT,
                                     var_vlrUnitario DECIMAL(9,2))
BEGIN
    DECLARE var_contador INT(11);

    SELECT COUNT(*)
    INTO var_contador
    FROM estoque e
    WHERE e.idProduto = var_idProduto;

    IF var_contador > 0 THEN
        UPDATE estoque e
        SET e.qtd = e.qtd + var_qtdComprada, e.vlrUnitario = var_vlrUnitario
        WHERE e.idProduto = var_idProduto;
    ELSE
        INSERT INTO estoque (idProduto, qtd, vlrUnitario)
        VALUES (var_idProduto, var_qtdComprada, var_vlrUnitario);
    END IF;
END $$
DELIMITER ;
```

Procedure: '**SP_AtualizaEstoque**'

Obs.: foi declarada uma variável **var_contador** para receber o valor da instrução SELECT COUNT(*)

Caso exista um **produto** cadastrado no **estoque** com o mesmo **var_idProduto** passado como parâmetro, então será inserido na variável **var_contador** o número de linhas que atendem a essa condição.

Procedure: '**SP_AtualizaEstoque**'

Posteriormente verifica-se o valor de **var_contador**, se for maior que 0 então executa-se um UPDATE na tabela '**estoque**', senão é feito um 'INSERT'.

Essa verificação pode ser feita de diversas maneiras, o programador pode implementar da melhor maneira possível.

Triggers – no MySQL

Será criada uma trigger para cada evento das tabelas **produtoEntrada** e **produtoSaida**

Nota: o MySQL não suporta múltiplos eventos em uma mesma trigger

Triggers – no MySQL

```
trg_produtoEntrada_AI
trg_produtoEntrada_AU
trg_produtoEntrada_AD

trg_produtoSaida_AI
trg_produtoSaida_AU
trg_produtoSaida_AD
```

TRIGGER trg_produtoEntrada_AI

Essa trigger será disparada após a inclusão de um registro na tabela de '**produtoEntrada**'.

```
-- Trigger trg_produtoEntrada_AI (INCLUSÃO de compra)
DELIMITER $$
CREATE TRIGGER trg_produtoEntrada_AI AFTER INSERT
ON produtoEntrada
FOR EACH ROW
BEGIN
  CALL SP_AtualizaEstoque (NEW.idProduto,
                           NEW.qtd,
                           NEW.vlrUnitario);
END $$
DELIMITER ;

-- TRIGGER de inclusão de compra (produtoEntrada)
-- Adiciona a quantidade comprada ao estoque.
```

TRIGGER trg_produtoEntrada_AD

Essa trigger será disparada após a exclusão de um registro na tabela de '**produtoEntrada**'.

```
-- Trigger trg_produtoEntrada_AD (EXCLUSÃO de compra)
DELIMITER $$
CREATE TRIGGER trg_produtoEntrada_AD AFTER DELETE
ON produtoEntrada
FOR EACH ROW
BEGIN
  CALL SP_AtualizaEstoque (OLD.idProduto,
                           OLD.qtd * - 1,
                           OLD.vlrUnitario);
END $$
DELIMITER ;

-- TRIGGER de exclusão de compra (produtoEntrada)
-- Remove a quantidade excluída do estoque (usa quantidade negativa).
```

TRIGGER trg_produtoEntrada_AU

Essa trigger será disparada após a atualização de um registro na tabela de '**produtoEntrada**'.

```
-- Trigger trg_produtoEntrada_AU (ALTERAÇÃO de compra)
DELIMITER $$
CREATE TRIGGER trg_produtoEntrada_AU AFTER UPDATE
ON produtoEntrada
FOR EACH ROW
BEGIN
  CALL SP_AtualizaEstoque (NEW.idProduto,
                           NEW.qtd - OLD.qtd,
                           NEW.vlrUnitario);
END $$
DELIMITER ;

-- TRIGGER de alteração de compra (produtoEntrada)
-- Atualiza o estoque com a diferença de quantidade.
```

TRIGGER trg_produtoSaida_AI

Essa trigger será disparada após a inserção de um registro na tabela de '**produtoSaida**'.

```
-- Trigger trg_produtoSaida_AI (INCLUSÃO de venda)
DELIMITER $$
CREATE TRIGGER trg_produtoSaida_AI AFTER INSERT
ON produtoSaida
FOR EACH ROW
BEGIN
  CALL SP_AtualizaEstoque (NEW.idProduto,
                           NEW.qtd * - 1,
                           NEW.vlrUnitario);
END $$
DELIMITER ;

-- TRIGGER de inclusão de venda (produtoSaida)
-- Reduz a quantidade do estoque.
```

TRIGGER trg_produtoSaida_AD

Essa trigger será disparada após a exclusão de um registro na tabela de '**produtoSaida**'.

```
-- Trigger trg_produtoSaida_AD (EXCLUSÃO de venda)
DELIMITER $$
CREATE TRIGGER trg_produtoSaida_AD AFTER DELETE
ON produtoSaida
FOR EACH ROW
BEGIN
  CALL SP_AtualizaEstoque (OLD.idProduto,
                           OLD.qtd,
                           OLD.vlrUnitario);
END $$
DELIMITER ;

-- TRIGGER de exclusão de venda (produtoSaida)
-- Recoloca no estoque a quantidade do produto excluído da venda.
```

TRIGGER trg_produtoSaida_AU

Essa trigger será disparada após a atualização de um registro na tabela '**produtoSaida**'.

```
-- Trigger trg_produtoSaida_AU (ALTERAÇÃO de venda)
DELIMITER $$
CREATE TRIGGER trg_produtoSaida_AU AFTER UPDATE
ON produtoSaida
FOR EACH ROW
BEGIN
  CALL SP_AtualizaEstoque (NEW.idProduto,
                           OLD.qtd - NEW.qtd,
                           NEW.vlrUnitario);
END $$
DELIMITER ;

-- TRIGGER de alteração de venda (produtoSaida)
-- Ajusta o estoque com base na diferença entre a quantidade antiga e a nova.
```

Observações

Algumas chamadas da *procedure* 'SP_AtualizaEstoque', antes de passar o parâmetro 'qtd' multiplicam esse valor por -1, essa operação muda o sinal matemático do valor para negativo.

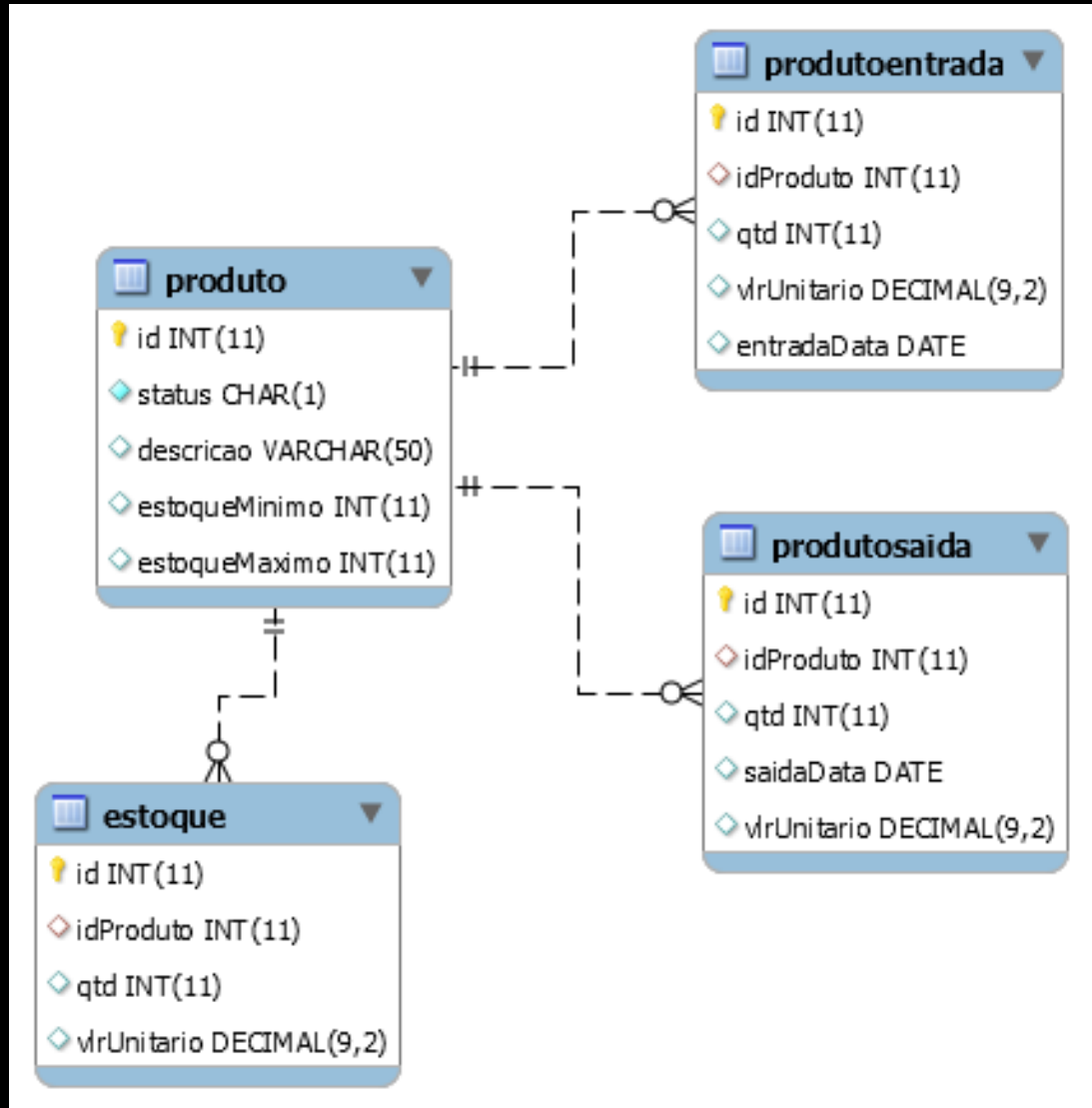
Dentro da *procedure* somamos as quantidades.

Quando passamos o sinal negativo ocorre uma subtração dos valores resultando em débito no estoque.

Fonte: http://www.devmedia.com.br/

No 'frigir dos ovos'

Se somarmos a quantidade em estoque com a quantidade vendida de um determinado produto, vamos obter a quantidade comprada.



Faça testes

```
-- Faça:

-- Testes de ENTRADA (compras)
INSERT em produtoEntrada
SELECT em produtoEntrada e SELECT em estoque
UPDATE em produtoEntrada
SELECT em produtoEntrada e SELECT em estoque
DELETE em produtoEntrada
SELECT em produtoEntrada e SELECT em estoque

-- Testes de SAÍDA (vendas)
INSERT em produtoSaida
SELECT em produtoSaida e SELECT em estoque
UPDATE em produtoSaida
SELECT em produtoSaida e SELECT em estoque
DELETE em produtoSaida
SELECT em produtoSaida e SELECT em estoque
```